What's New: Finding Significant Differences in Network Data Streams

Graham Cormode
DIMACS, Rutgers University
graham@dimacs.rutgers.edu

S. Muthukrishnan
DCIS, Rutgers University
muthu@cs.rutgers.edu

Abstract— **Monitoring and analyzing network traffic usage patterns is vital for managing IP Networks. An important problem is to provide network managers with information about changes in traffic, informing them about "what's new". Specifically, we focus on the challenge of finding significantly large differences in traffic: over time, between interfaces and between routers. We introduce the idea of a *deltoid*: an item that has a large difference, whether the difference is *absolute*, *relative* or *variational*.**

We present novel algorithms for finding the most significant deltoids in high speed traffic data, and prove that they use small space, very small time per update, and are guaranteed to find significant deltoids with pre-specified accuracy. In experimental evaluation with real network traffic, our algorithms perform well and recover almost all deltoids. This is the first work to provide solutions capable of working over the data with one pass, at network traffic speeds.

Keywords: Network measurements. Traffic data analysis.

I. Introduction

IP networks are sophisticated engineering systems. Monitoring and analyzing network traffic usage patterns is essential for managing these systems. For example, *provisioning* IP networks needs capacity planning and forecasting which needs detailed analysis of traffic usage over time. Running a service—hosting, providing network connectivity, etc—needs detailed *accounting* for billing, verifying Service Level Agreements, periodic reporting of usage per customer, etc. Enforcing and ensuring the *security* of the infrastructure needs constant monitoring of network activity for patterns of anomalous traffic. In general, it is a fundamental operational detail in interacting with an IP network at any level—single user or a large ISP—that one have tools to gather and analyze traffic usage.

Our study here is primarily motivated by analysis of massive, high speed data generated by IP networks, from the perspective of a large ISP. For motivation, consider analysis of the header information on each IP packet, or at a higher level of aggregation, the records of IP flows say from Cisco's netflow, from each of the routing elements of a large ISP. Our focus is on *near-real time* analysis such as warranted by network monitoring scenarios. In this context, there are two key questions.

What are the performance constraints for high speed network data analysis? Capturing per packet information or netflow records for each router and transporting it to data warehouses is unrealistic, because of the storage costs as well as the transportation overhead. A back-of-the-envelope calculation with even 10's of OC48's such as those found in a large ISP backbone will illustrate this fact. Unlike in telephone networks where billing "per record/call" is (has been) the norm and is subject to legal requirements, IP network operators have less motivation to collect or archive packet or flow records since it does not have a direct and immediate impact on revenue, and it is not mandated. Instead, a more realistic scenario is to collect some aggregated information or monitor specific "queries" of interest on the traffic stream. That entails performing computations per packet or per netflow record, at the router itself or at collection boxes associated with the routers. This in turn presents well known performance bottlenecks: one needs methods that will (1) use small amount of memory because memory such as SRAM with access time commensurate with IP traffic is expensive and it is impractical to attach large memory of this caliber to each interface card of typical routers in large ISPs and (2) use very few memory accesses per packet or flow record.

Both these constraints are well known in networking community, and have been articulated in the classical context of packet switching, more recently in packet classification and IP lookups, and in the emerging context of monitoring high speed traffic data (see [16] for a short, but excellent overview of these constraints). Our algorithmic results in this paper are designed with these performance criteria in mind.

What are specific data analyses of interest to monitoring high speed traffic data? Typically, the focus is on monitoring a few simple aggregates that will serve as "signals" for ongoing phenomenon. For example, one may monitor the number of distinct "flows"—distinct source IP addresses, or distinct TCP connections, etc—ongoing in a link: steep increases in this number may correlate with certain attacks or port scans [17], [32]. In a similar spirit, one may wish to calculate the number of "tiny" flows, that is, the ones that involve few packets only [12]. Another example is to monitor "heavy hitters", i.e., those flows that represent a significantly large proportion of the ongoing traffic or the capacity of the link [15].

In this paper, we study a somewhat related class of problems of finding entities—addresses, flows eg., comprising source/destination IP addresses and port numbers or combinations thereof, etc—that *differ significantly* in traffic level from one time window to another, from one interface to other, or from one router to another. The traffic level may be counted in

terms of the number of connections, packets, bytes, etc. These are therefore heavy hitters in the *difference* of traffic levels either across time, interface or routes. Currently, network managers tell us that they look for significant differences in the traffic levels while operating a network; monitoring significant differences is an intuitively powerful way to summarize the changes in the network, and therefore, draw human attention to these across the network over time. What is needed is a way to highlight the things that are different, that is, to find "what's new" between different traffic streams.

Our main results are extremely efficient methods that work on high speed IP traffic data and detect significantly large differences in traffic levels across time and network elements. Our contributions are:

- We formalize intuitive notions of "differences"—*deltoids*, as we call them, including absolute, relative or variational deltoids—and initiate the study of methods for finding significantly large deltoids between high speed IP network data streams.
- We design efficient algorithms for finding significant deltoids on high speed data. We analytically *prove* that they (a) use small space, (b) take small time per packet or flow record update, and (c) find significant deltoids within pre-specified accuracy quickly. The algorithms use novel group tests in a combinatorial group testing framework that underlies all the algorithms, and work without any assumption on the input data source and give the first solution to these problems.
- We implement and test our algorithms on various real life network data—netflow and SNMP data from IP networks as well as telephone records—and show that deltoids are interesting. Even without engineering our algorithm using standard techniques of parallelization, hardware implementation or exploiting the special structure of IP data, our algorithms can process a million records a second on a cheap desktop computer. Thus our solutions appear well suited for high speed network monitoring applications.

Our work lies in the intersection of research in many communities. The networking community has recently started studying what traffic data analysis problems can be solved at line speed: finding heavy hitters [11], [15], [24], counting distinct flows [17], etc. Our work here extends this list by providing methods to do quite nontrivial analyses such as what are the significant differences in traffic levels across network elements and time. The precise problem of looking for deltoids is implicit in the exploratory approach inherent in network management. The problem of detecting relative deltoids for example has been posed as an open problem in [6], [22], and has been often stated in informal discussions with researchers and network operators. Finally, our work is grounded in Algorithms research. The combinatorial group testing approach for stream computations has been developed in [10], [20], [21]. Most similar is [10] for the problem of finding heavy hitters. Here, we expand previous work substantially, in particular, by designing novel tests that work for different deltoids. This not only gives us the first known theoretical results we derive in this paper, but also serves to position the combinatorial group testing as a general framework for detecting significant entities, be they volumes or differences.

Map. In Section II, we present a discussion of what various differences of interest are, and formally define the problem of finding significant deltoids, We also show why standard methods such as sampling do not work. In Section III and IV we present our group-testing framework, and present specific results for different deltoids. In Section V, we present our experimental results with real network data. Some extensions are described in Section VI and in Section VII, we discuss related work. Concluding remarks are made in Section VIII.

II. Preliminaries

A. Difference Detection Problems: Informal Discussion

We focus on finding items which exhibit large difference. We call such items "deltoids", to denote items whose difference, or delta, is noteworthy. There could be many possible ways to measure how individual items have changed. We illustrate these by considering the number packets sent by a particular IP address through a given interface aggregated by hour.

1) *Absolute Difference:* A large difference between the number of packets sent in one hour and the next.
2) *Relative Difference:* A large ratio of the number of packets sent in one hour and the next.
3) *Variational Difference:* A large variance of the number of packets taken over multiple time periods.

We must be precise about what is meant by "large", since it will depend on the volume of traffic, and whether we are counting packets, bytes or flows. More than this, it also depends on the distribution of traffic itself: if we look for deltoids between traffic going into and coming out of a link, then a difference of a few packets would be significant, whereas between the traffic going onto the link in one hour and the next, the difference would have to be much larger to be noteworthy. It is therefore vital that the notion of a significant difference is *traffic dependent*.

Our solution is to look for differences which are a user-specified fraction of the *total difference*. That is, items whose difference is some fraction, say 1% or 5% of the sum of differences of all items. Given a fraction ϕ, there can be at most $1/\phi$ deltoids for any notion of difference, although this bound is unlikely to be reached by realistic traffic, since we expect much of the difference to be from non-deltoids.

Each of our notions of difference captures a different situation. A busy web server such as CNN.com will experience a notable absolute difference in its traffic while an exciting news story is unfolding. Flash crowds, or "the slashdot effect" will result in a large relative difference for a server that normally experiences lower traffic (another relative difference would be if the server crashes under the increased load and its outbound traffic falls from high to zero). Meanwhile, high variance

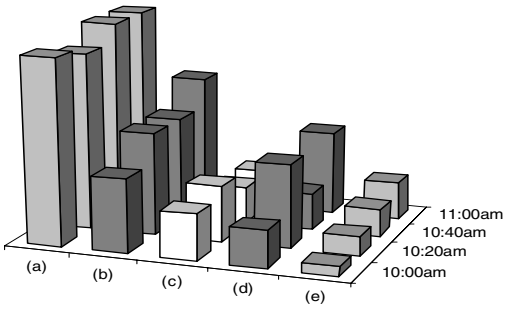


Fig. 1. Items displaying different kinds of difference: (b) has the highest absolute difference between 10am and 11am, (e) has the highest relative difference, and (d) has the highest variance.

detects items whose traffic is variable over time, such as office networks, whose traffic will be high during working hours, and low overnight. These notions can be distinct: Figure 1 depicts a situation with five different items (a)—(e) and their values over equal time periods between 10am and 11am. The items with highest relative and absolute difference between the first and last reading, and with the highest variance are all distinct (and distinct from the item with highest overall count).

Because of the potentially high volume of network traffic and high link speeds, any method devised for finding deltoids needs to provide truly high performance in order to be considered for deployment in real network monitoring situations. See [16] for a nice discussion of the rationale. We summarize the requirements:

- **Fast Update Speed.** Solutions have to be capable of operating at network line speed on a per packet or per flow record basis. Thus per packet or per flow record processing has to be very fast so that data processing is carried out in real time. This rate varies greatly, depending on the capacity of the link and the nature of the traffic. IP traffic on fast backbone links can be many millions of packets per second but other situations generate traffic at much lower rates.
- **Low Space Requirement.** Although memory and disk is increasingly cheap and plentiful, storing and processing traffic data at per packet or per netflow record speed calls for high speed memory with very small access times. Such memory (such as SRAMs) can be expensive, and so it is desirable that solutions use small space for processing and storing summaries of IP traffic data streams.
- **Efficient, Accurate Queries.** The operation of recovering the deltoids should not be a costly one. Although this operation can be done offline, and so does not have the same time restrictions as the update operation, still it should be relatively fast to find the deltoids. It is very important that the operation should also give *guarantees* about the accuracy of its output.

B. Standard Approaches

Meeting all these requirements is not straightforward. Many natural first ideas fail on one or more of these criteria. We briefly discuss various simple attempts to solve this problem, and explain their shortcomings.

- **Sampling.** Reducing the storage cost by sampling and storing, some very small fraction—say 1% or less—has the disadvantage that we are likely to miss important information about deltoids. To achieve a reasonable amount of storage space, the rate of sampling will have to be very low, thereby missing many packets or netflow records. In the worst case, deltoids are missed entirely by the sampling and so cannot be recovered.
- **Heavy Hitters.** Several methods have been published recently for finding the "heavy hitter" items, which are those whose traffic is above some threshold of the total traffic [10], [15], [27]. This is a related notion to deltoids, since heavy hitters are a special case of deltoids: the deltoids found between a traffic stream and an empty stream are precisely the heavy hitters. So this suggests the following solution: for each stream, find and store the heavy hitters which account for more than ϕ of the total traffic. Then given two streams, output as the deltoids all items which are heavy hitters in one stream but not the other. Such an approach is unfortunately severely flawed. For example, the heavy hitters might be identical in both streams: some items are always popular (such as popular websites). Because deltoids are defined in relation to the *sum of the differences* instead of the *sum of the traffic*, then it is possible that no deltoids are heavy hitters, and so this method will not find any of the true deltoids, and will output items that are not deltoids. In our experiments, we found that this heuristic performed generally poorly.
- **Sketch-based Methods.** Sketches are a class of powerful, small space approximations of distributions [4]. It is possible to create sketches for each stream so that combining sketches for multiple streams allows the (absolute) difference for each item to be found [20], [25]. However, this approach suffers a major drawback: in order to compare two streams, we must somehow know which items to query in order to find those with large change. So, either we must query every item (eg all 2^{32} IP addresses), or we use the current stream as a sequence of items to test: for every address that is seen, test whether this has a large change. This approach is problematic, since it requires extra processing: we must update our data structure and test it for every item in the stream. It also means we must decide in advance what comparison to do, and does not allow the more flexible approach of comparing arbitrary pairs of streams after the stream has been seen. The main point is that a sketch is a small storage approximation of the data, but if we want to use that to find deltoids, we must exhaustively cycle through many possible values.

C. Problem Formulation

We will consider streams $S_1, S_2, \ldots S_m$ which represent the data of interest over collected over fixed time periods, eg. each stream represents observed traffic flows from a particular hour or day. These can be thought of as defining vectors, where the ith entry of the vector for S_j represents the quantity associated with item i after processing the whole of the jth stream. We shall use S_j to refer to both the stream, and also the implicit vector that it defines, and so $S_j[i]$ denotes the total for item i in the jth stream. The dimension of S_j is n, meaning that $i \in \{0 \ldots n-1\}$.

For example, the streams might represent flow volume from each source IP address on a given link, one stream per hour. Then $n = 2^{32}$ and $S_j[i]$ represents the total flow volume from source IP address i in the jth hour. Streams of IP data are modeled by the *cash register* model of streams [29]: this means that the same item i can be observed multiple times in the stream, and each contribution adds to the value of $S_j[i]$. This naturally describes a stream of IP packets: each packet has an address i, and a packet size p so that $S_j[i] \leftarrow S_j[i] + p$. The challenges here are multiple: first, to process the streams as they arrive in real time, at network line speeds; and second, to obtain a concise, approximate representation of each stream, so that we use much less fast memory than we would to represent S_j exactly. Then, given queries of the form (j, k), we want to find particular items i which behave differently in S_j than in S_k.

We can now formalize the idea of deltoids.

- *Absolute Difference.* The *absolute difference* of an item i is $|S_j[i] - S_k[i]|$.
- *Relative Difference.* The *relative difference* of an item i is $S_j[i]/\max\{S_k[i], 1\}$.[1]
- *Variational Difference.* The *variational difference* (variance) of an item i over ℓ streams is given by $\sum_{j=1}^{\ell}(S_j[i] - \sum_{k=1}^{\ell} S_k[i]/\ell)^2$.

We shall describe methods to find items whose absolute, relative or variational difference is high. We use the term *deltoid* to refer to an item whose difference is large relative to the total differences.

Definition 1 (Exact Deltoids): For any item i, let $D[i]$ denote the difference of that item, be it absolute, relative or variational difference. A ϕ-deltoid is an item i so that $D[i] > \phi \sum_x D[x]$.

Our solutions rely on a slight relaxation of the problem, where we talk of approximate deltoids.

Definition 2 (Approximate Deltoids): Given $\epsilon \leq \phi$, the ϵ-approximate ϕ-deltoid problem is to find all items i whose difference $D[i]$ satisfies $D[i] > (\phi+\epsilon)\sum_x D[x]$, and to report no items where $D[i] < (\phi-\epsilon)\sum_x D[x]$. Items between these thresholds may or may not be output. We consider the set of deltoids, denoted $Deltoids$, defined as

$$i \in Deltoids \Rightarrow D[i] > (\phi+\epsilon)\sum_x D[x]$$
$$i \notin Deltoids \Rightarrow D[i] < (\phi-\epsilon)\sum_x D[x].$$

[1]The 1 term makes sure there is no 0 in the denominator.

All our algorithms are probabilistic with user-defined parameter δ which is the upper bound on the probability of the algorithm failing. All our bounds will involve parameters ϕ, ϵ and δ, in addition to n. We will assume each $S_j[i]$ can be stored in one computer word, as is standard. All the space bounds we state below are in terms of the number of words.

III. Algorithmic Framework

Our solutions are based on Group Testing. The underlying principle is to make use of *tests* which, given a subset, or *group* of items, tell us whether there is a deltoid within the group. In general, the test may err with some probability, and so we will need to bound the chance of false positives (including an item which is not a deltoid in the output) and false negatives (failing to include a deltoid in the output).

Our Non-Adaptive Group Testing Procedure is divided into two parts: *identification*, to find a set of "candidates" which should include all deltoids, and *verification*, which removes items from the set of candidate items which are not deltoids. For each part we keep a data structure, which consists of sets of "Test" data structures: as items arrive, they are included in appropriate test data structures, as described below. All our procedures will use essentially the same structure of groups; what will vary is the tests that are used. We will first describe this structure and how it is used. In the next section, we will describe how to make tests for deltoids.

A. Identification

Group Structure. The groups are subsets of the items, defined by pairwise independent hash functions [28]. Given approximation factor ϵ and failure probability δ, choose $t_{id} = \log \frac{1}{\delta}$ such hash functions $h_{1\ldots \log \frac{1}{\delta}} : \{0 \ldots n-1\} \rightarrow \{1 \ldots g\}$. Here g is the number of groups, to be specified later. Set $G_{a,b} = \{i | h_a(i) = b\}$.

Tests. Within each group keep $1 + \log n$ data structures $T_{a,b,c}$ which allow us to pose tests on the items in the group. The data structure will depend on the nature of the difference we are trying to detect and will be specified later; for now, assume that each test reports whether there is a deltoid in the group. Let B_c denote the set of integers whose binary representation has a 1 in the cth bit position for $c = 1 \ldots \log n$; for convenience of notation, let $B_0 = \{0 \ldots n-1\}$. Then $T_{a,b,c}$ applies to all items in $G_{a,b} \cap B_c$. We will assume that the tests here are *linear*: that is, the tests are a linear function[2] of the data. Let $T'_{a,b,c}$ denote the complement of $T_{a,b,c}$: $T_{a,b,c}$ reports whether there is a deltoid in $G_{a,b} \cap B_c$, and $T'_{a,b,c}$ reports whether there is a deltoid in $G_{a,b} \backslash B_c$. By linearity of the test function, $T'_{a,b,c} = T_{a,b,0} - T_{a,b,c}$. Finally, for some test T, let $|T|$ denote the outcome of the test: $|T| = 1$ means that the test returned positive, and $|T| = 0$ otherwise.

Group Testing for Identification. In order to find the deltoids between S_j and S_k, we will need to combine the test data

[2]f is a linear function if it satisfies $f(x+y) = f(x) + f(y)$ and $f(ax) = af(x)$ for all x and y in the domain of the function, and for all scalars a.

structures for each stream, $T_{a,b,c}(j)$ and $T_{a,b,c}(k)$ to get $T_{a,b,c}(j,k)$. How this achieved will vary from test to test; from now on, we will treat the tests as black box objects which report whether there is a deltoid within the subset that the test is operating on. We then apply the following procedure:

- For each group $G_{a,b}$, if $|T_{a,b,0}(j,k)| = 0$, conclude that there is no deltoid in the group, and go to the next group.
- Otherwise, we use the results of the other tests for that group to identify the deltoid. For each value of c, if $|T_{a,b,c}| = |T'_{a,b,c}|$ then either both are negative, and there is no deltoid in the group after all, or both are positive, and there are two or more deltoids in the same group. In both these cases, we reject the group $G_{a,b}$.
- Otherwise, if $|T_{a,b,c}| = 1$ then the deltoid $i \in B_c$ so it has a 1 in the cth bit position; else $|T'_{a.b.c}| = 1$ and so i has 0 in the c bit position. So the full binary representation of i can be recovered.
- If the group is not rejected, then some item i is found and so it is added to the set of *candidate items*, which are believed to be ϵ-approximate ϕ-deltoids.

B. Verification

In practice, the tests will not be perfect, but will themselves have some probability of failure, which can lead to false positives. Some simple checks can be made to avoid this. Having found an item $i \in G_{a,b}$ which is believed to be a deltoid, a first "sanity check" is to check that $h_a(i) = b$: if not, then clearly the tests erred, and the item should be rejected. To give good guarantees about the items that are found, we will additionally keep a Verification data structure. This closely resembles the Identification structure, but is constructed with different parameters.

Groups and Tests. Using the same parameters ϵ and δ, we choose $t_{ver} = 4 \log \frac{1}{\delta}$ new hash functions $f_1 \ldots f_{4 \log 1/\delta} : \{1 \ldots n\} \rightarrow \{1 \ldots v\}$ from a family of pairwise independent hash functions This time we keep just a single test data structure for each group, $V_{a,b}$. The tests and number of groups are chosen so the probability that each test errs is at most $\frac{1}{8}$.

Group Testing for Verification. For each candidate item, compute the groups that it falls into in the verification data structure. For each of these groups, compute the test outcome, and take the majority vote of these tests (positive or negative) as the overall result for the item. If the item is positive, include it in the output as an ϵ-approximate ϕ-deltoid.

Theorem 1: If the probability that each test $V_{a,b}$ gives the wrong answer is at most $\frac{1}{8}$, then

$$i \in Deltoids \Rightarrow \mathsf{Pr}[(\textstyle\sum_{a=1}^{4 \log \frac{1}{\delta}} |V_{a,f(i)}|) < 2 \log \frac{1}{\delta}] < \delta$$
$$i \notin Deltoids \Rightarrow \mathsf{Pr}[(\textstyle\sum_{a=1}^{4 \log \frac{1}{\delta}} |V_{a,f(i)}|) \geq 2 \log \frac{1}{\delta}] < \delta$$

Proof:
$i \notin Deltoids \Rightarrow \mathsf{E}((\sum_{a=1}^{4 \log \frac{1}{\delta}} |V_{a,f(i)}|) = \frac{1}{2} \log \frac{1}{\delta}$. So $2 \log \frac{1}{\delta} = 4\mathsf{E}((\sum_{a=1}^{4 \log \frac{1}{\delta}} |V_{a,f(i)}|)$. By Chernoff bounds [28], $\mathsf{Pr}[(\sum_{a=1}^{4 \log \frac{1}{\delta}} |V_{a,f(i)}|) \geq 2 \log \frac{1}{\delta}] < \delta$. The other case is symmetric. ∎

Consequently, given this set up we can be sure that each item passed by the identification and verification stages has probability at most δ of not being a deltoid. Similarly, every deltoid has probability δ of not being passed by the verification stage. We are free to set δ to be arbitrarily small, and the dependence on δ is fairly weak.

Choosing a Threshold. We must choose the threshold for an item being a deltoid. Each of the tests that will introduce will involve comparing a numeric quantity to $\phi \sum_i D[i]$, a fraction of the total difference. So in particular we need to know $\sum_i D[i]$ to be able to make the test. For each test, we will show how to find this quantity exactly, or give a good approximation.

Complete Update Procedure. The full update procedure for the Combinatorial Group Testing is

- Read new item i with traffic p (bytes, packets or flows).
- For $a = 1$ to t_{id} do
  - For $c = 0$ to $\log n$ do
    - If $i \in B_c$, update $T_{a,h_a(i),c}$ with p
- For $a = 1$ to t_{ver} do
  - Update $V_{a,f_a(i)}$ with p

The running time is therefore $O(t_{ver} + t_{id} \log n)$ test updates per item in the stream. For tests which take constant time to update (as is the case for all tests we consider here), then this cost is $O(\log(n) \log \frac{1}{\delta})$. This meets our requirement of being fast to update. For each kind of deltoid, we will additionally show that the overall space requirements are also low.

IV. Finding Deltoids

A. Absolute Deltoids

For absolute deltoids, each test data structure is simply a single variable,

$$T_{a,b,c} = \textstyle\sum_{i \in G_{a,b} \cap B_c} S_j[i].$$

This data structure is clearly linear and straightforward to maintain under updates: when an update to item i of p arrives, just add p to all counters $T_{a,h_a(i),c}$. We define the combination of test for streams j and k as:

$$T_{a,b,c}(j,k) = |T_{a,b,c}(j) - T_{a,b,c}(k)|$$
$$|T_{a,b,c}(j,k)| = 1 \iff T_{a,b,c}(j,k) > \phi ||S_j - S_k||_1.$$

We set the number of groups in the identification procedure to be $g = \frac{2}{\epsilon}$ and for the verification $v = \frac{8}{\epsilon}$. The following lemma shows that we have constant probability of finding each deltoid in each group that it is counted in.

Lemma 1: $i \in Deltoids \Rightarrow$
$$\forall a : \mathsf{Pr}[\forall c : \; (|T_{a,h_a(i),c}(j,k)| = 1 \iff i \in B_c) \wedge (|T'_{a,h_a(i),c}(j,k)| = 1 \iff i \notin B_c)] \geq \tfrac{1}{2}$$

Proof: If $i \in Deltoids$, suppose for some a $|T_{a,b,0}(j,k)| = 0$ (a false negative), then (by definition)

$$|D[i]| > (\phi + \epsilon)||S_j - S_k||_1 \wedge |\sum_{x \in G_{a,b}} D[x]| < \phi ||S_j - S_k||_1$$

$$\text{So} \quad |T_{a,b,0}(j,k)| = 0 \Rightarrow |\sum_{x \neq i, x \in G_{a,b}} D[x]| > \epsilon ||S_j - S_k||_1.$$

Let the random variable[3] $X_i = \sum_{x \neq i, x \in G_{a,b}} |D[x]|$. For all c, $i \in Deltoids \wedge X_i < \epsilon ||S_j - S_k||_1 \Rightarrow$

$$\begin{array}{ll} i \in B_c \Rightarrow T_{a,b,c}(j,k) \geq D[i] - X_i & > \phi ||S_j - S_k||_1) \wedge \\ i \notin B_c \Rightarrow T_{a,b,c}(j,k) \leq X_i & < \phi ||S_j - S_k||_1) \end{array}$$

$$\Rightarrow |T_{a,b,c}(j,k)| = 1 \iff i \in B_c;$$

the case for $i \notin B_c$ is symmetric with T' replacing T. Since $|\sum_{x \in G_{a,b}} D[x]| \leq X_i$:

$$\mathsf{Pr}[|\sum_{x \in G_{a,b}} D[x]| < \epsilon ||S_j - S_k||_1] \geq \mathsf{Pr}[X_i < \epsilon ||S_j - S_k||_1].$$

By the pairwise independence of the hash functions, $\mathsf{E}(X_i) = \frac{\epsilon}{2}||S_j - S_k||_1$, and by the Markov inequality,

$$\mathsf{Pr}[X_i < \epsilon ||S_j - S_k||_1] = \mathsf{Pr}[X_i < 2\mathsf{E}(X_i)] > \frac{1}{2}.$$

■

This means that for each Identification group that each deltoid falls in, there is a constant probability that all tests give the correct output, and so consequently we can identify it. Since each deltoid falls in $\log \frac{1}{\delta}$ groups, then the probability that it is not detected in any of them is less than $2^{\frac{1}{\delta}} = \delta$, so the probability that it *is* found is at least $1 - \delta$.

Lemma 2: $i \in Deltoids \Rightarrow \mathsf{Pr}[|V_{a,f_a(i)} = 0] < \frac{1}{8} \wedge\ i \notin Deltoids \Rightarrow \mathsf{Pr}[|V_{a,f_a(i)} = 1] < \frac{1}{8}$

Proof: The proof is similar to the previous lemma:

$$\begin{array}{rll} i \in Deltoids \Rightarrow & \mathsf{Pr}[|V_{a,f_a(i)}(j,k)| = 0] & \\ \leq & \mathsf{Pr}[\sum_{x \neq i, x \in G_{a,f_a(i)}} |D[x]| > \epsilon ||S_j - S_k||_1] & \\ < & \frac{1}{8} \text{ by the same argument as above.} & \\ i \notin Deltoids \Rightarrow & |D[i]| < (\phi - \epsilon)||S_j - S_k||_1 & \\ \Rightarrow & \mathsf{Pr}[|V_{a,f_a(i)}(j,k)| = 1] & \\ \leq & \mathsf{Pr}[\sum_{x \neq i, x \in G_{a,f_a(i)}} |D[x]| > \epsilon ||S_j - S_k||_1] & \\ < & \frac{1}{8} & \end{array}$$

■

So the probability of each test erring is at most $\frac{1}{8}$, and applying Theorem 1 gives the result that each deltoid is passed by the Verification stage with probability $1 - \delta$ while non-deltoids are rejected with probability $1 - \delta$.

Setting a Threshold. To set the threshold for searching for absolute difference deltoids, we need to compute $||S_j - S_k||_1$. This can be accomplished by keeping an additional "sketch" structure for each stream and combining them to make a good quality approximation of the L_1 difference of the two streams. Such techniques are well documented in the literature for example in [18], [23].

Theorem 2: Each ϵ-approximate absolute deltoid is found by the above algorithm with probability at least $1 - \delta$. The space required for finding absolute difference deltoids is $O(\frac{1}{\epsilon}\log(n)\log\frac{1}{\delta})$. The time to update the tests is $O(\log(n)\log\frac{1}{\delta})$ per item in the stream, and the expected time to find deltoids is $O(\frac{1}{\epsilon}\log(n)\log\frac{1}{\delta})$.

[3]This is random variable depending on the random choice of the hash function h from the set of pairwise independent hash functions.

B. Variational Deltoids

To find items with the highest variational difference, we first describe how to build a test for finding items which are large in their squares, and then show how to adapt this to finding high variance items. The test construction for variations is more complex, and is based on the "sketch" described by Alon, Matias and Szegedy [4]. Naively, we could keep a full sketch for each test, but this would blow up the space and time to process each item. Here, we show that keeping a single counter for each test is sufficient to find deltoids. We also crucially rely on the linearity properties of the tests derived from these sketches. For each hash function $h_a : \{0 \ldots n-1\} \rightarrow \{1 \ldots \frac{d}{\epsilon^2}\}$ which divides items into groups (with d to be specified later), we additionally keep a second hash function z_a which is drawn from a family of 4-wise independent hash functions mapping the items $\{0 \ldots n-1\}$ uniformly onto $\{+1, -1\}$. For each group, compute

$$T_{a,b,c} = \sum_{i \in G_{a,b} \cap B_c} S_j[i] z_a(i).$$

Again, this is easy to maintain this when S_j is presented as an unaggregated stream of values, since for each update we just have to add the update multiplied by $z_a(i)$ onto $T_{a,h_a i,c}$ for all values of a and c.

Lemma 3: For each group that item i falls in, $T^2_{a,b,c}$ is a good estimate for $S_j[i]^2$: with probability at least $\frac{2d}{(d-1)^2}$, then $|T^2_{a,b,c} - S_j[i]^2| < \epsilon ||S_j||^2$.

Proof: The analysis proceeds in a similar way to that given in [4]. Let $I_{i,x}$ be an indicator variable so that $I_{i,x} = 1 \iff (h_a(x) = h_a(i))$, and 0 otherwise. By the independence properties of z_a, the expectation $\mathsf{E}(z_a(i)z_a(x)) = 0$ for $i \neq x$, and is always 1 otherwise. The expectation of $T^2_{a,b,c}$ is

$$\begin{array}{rl} \mathsf{E}(T^2_{a,b,c}) = & (S_j[i]z_a(i) + \sum_{x \neq i} I_{i,x} S_j[x] z_a(x))^2 \\ = & S_j[i]^2 + \sum_{x \neq i} \mathsf{E}(2 S_j[i] z_a(i) I_{i,x} S_j[x] z_a(x)) + \\ & \sum_{x,y \neq i} \mathsf{E}(z_a(x) z_a(y) I_{i,x} I_{i,y} S_j[x] S_j[y]) \\ \leq & S_j[i]^2 + 0 + \frac{\epsilon^2}{d}||S_j||_2^2 \\ = & S_j[i]^2 + \frac{\epsilon^2}{d}||S_j||_2^2 \end{array}$$

Similarly, the variance, $\mathsf{Var}(T^2_{a,b,c})$ is at most $\frac{2\epsilon^2}{d}||S_j||_2^4$. Consider the probability that $T^2_{a,b,c}$ is not a good estimate of $S_j[i]^2$: then the difference between the two quantities is more than $\epsilon||S_j||_2^2$. By applying the Chebyshev inequality, and the fact that $\epsilon < 1$:

$$\begin{array}{rl} & \mathsf{Pr}[|T^2_{a,b,c} - S_j[i]^2| > \epsilon ||S_j||_2^2] \\ \leq & \mathsf{Pr}[|T^2_{a,b,c} - S_j[i]^2 - \frac{\epsilon^2}{d}|S_j||_2^2| > \epsilon(1 - \frac{\epsilon}{d})||S_j||_2^2|] \\ < & \mathsf{Pr}[|T^2_{a,b,c} - \mathsf{E}(T^2_{a,b,c})| > \frac{(d-1)\epsilon}{d}||S_j||_2^2] \\ < & \frac{d^2 \mathsf{Var}(T^2_{a,b,0})}{(d-1)^2 \epsilon^2 ||S_j[i]||_2^4} = \frac{2d}{(d-1)^2} \end{array}$$

■

The condition for Variational Deltoids can be re-written in terms of sums of squares. The contribution to the variance of item i from the ℓ streams is given by

$$\sigma^2[i] = \sum_{j=1}^{\ell} (S_j[i] - \mu[i])^2; \quad \mu[i] = \sum_{k=1}^{\ell} \frac{S_k[i]}{\ell}$$

By the linearity of the test function then, we can compute a single estimate for $\sigma^2(j)[i]$ as $(T_{a,b,c}(j) - \sum_{k=1}^{\ell} T_{a,b,c}(k)/\ell)^2$. Denote the total variance of all items, $\sum_i \sigma^2[i]$ as $\sigma^2(\ell)$.

For the test for variational deltoids, set $g = \frac{6}{\epsilon^2}$, $v = \frac{18}{\epsilon^2}$, and

$$T_{a,b,c}(\ell) = \sum_{j=1}^{\ell} \left(T_{a,b,c}(j) - \sum_{k=1}^{\ell} \frac{T_{a,b,c}(k)}{\ell} \right)^2$$

$$|T_{a,b,c}(\ell)| = 1 \iff T_{a,b,c}(\ell) > \phi\sigma^2(\ell).$$

Lemma 4: $i \in Deltoids \Rightarrow$
$\forall a, c : \mathsf{Pr}[\quad (|T_{a,h_a(i),c}(j,k)| = 1 \iff i \in B_c)$
$\wedge \quad (|T'_{a,h_a(i),c}(j,k)| = 1 \iff i \notin B_c)] > \frac{1}{2}$

Proof: For $i \in Deltoids$ and any a and c, let $b = h_a(i)$. Assume that $i \in B_c$ (the other case is symmetric). Hence, by Lemma 3 with $d = 6$ and using linearity of expectation with the fact the variance of the estimator is bounded by the sum of the variances, then

$$\mathsf{Pr}[|T_{a,b,c}(\ell) - \sigma^2[i]| > \epsilon\sigma^2(\ell)] < \frac{12}{25} < \frac{1}{2}.$$

Since $i \in Deltoids \Rightarrow \sigma^2[i] \geq (\phi + \epsilon)\sigma^2(\ell)$, then

$$\begin{aligned} i \in B_c \Rightarrow \quad & \mathsf{Pr}[|T_{a,b,c}(\ell)| = 1] = \mathsf{Pr}[T_{a,b,c}(\ell) \geq \phi\sigma^2(\ell)] \\ \leq \quad & \mathsf{Pr}[|T_{a,b,c}(\ell) - (\phi+\epsilon)\sigma^2(\ell)| \geq \epsilon\sigma^2(\ell)] < \tfrac{1}{2} \end{aligned}$$

using the above. For the other direction, assume $i \notin B_c$. Since i is not in this group, then effectively $\sigma^2[i] = 0$ for this group. Then by Lemma 3 again, and using the fact that $\epsilon \leq \phi$, it follows that

$$\begin{aligned} i \notin B_c \Rightarrow \quad & \mathsf{Pr}[|T_{a,b,c}(\ell)| = 1] = \mathsf{Pr}[T_{a,b,c}(\ell) \geq \phi\sigma^2(\ell)] \\ \leq \quad & \mathsf{Pr}[|T_{a,b,c}(\ell) - 0| \geq \epsilon\sigma^2(\ell)] < \tfrac{1}{2} \end{aligned}$$

■

Hence, amplifying the probability by $\log \frac{1}{\delta}$ repetitions, there is probability at least $1 - \delta$ that each deltoid will be found.

The probability of failure of the Verification tests is less than $\frac{1}{8}$, by again observing that the expectation of each verification test is a function of the variance of the item i, $\sigma^2[i]$, and by substituting $d = 18$ into Lemma 3, giving the probability of a good estimate to be $\frac{36}{172} < \frac{1}{8}$. So deltoids pass *all* Verification tests with probability at least $1 - \delta$, while non-deltoids are passed with probability at most δ.

Computing a Threshold. In order to set the threshold based on $\phi\sigma^2$, we need to know σ^2 itself. This can be done by making an appropriate sketch data structure, but it turns out that the data structure that we want to make is precisely that of the Verification tests: an unbiased estimator for σ^2 is

$$\text{median}_a \sum_b \left(\sum_{j=1}^{\ell} \left(V_{a,b}(j) - \sum_{k=1}^{\ell} \frac{V_{a,b}(k)}{\ell} \right)^2 \right).$$

Theorem 3: Each ϵ-approximate variational deltoid is found by the above algorithm with probability at least $1 - \delta$. The space required for finding variational deltoids is $O(\frac{1}{\epsilon^2}\log(n)\log\frac{1}{\delta})$. The time to update the tests is $O(\log(n)\log\frac{1}{\delta})$ per item in the stream, and the expected time to find deltoids is $O(\frac{1}{\epsilon^2}\log(n)\ell\log\frac{1}{\delta})$.

C. Relative Deltoids

Finding relative deltoids gives an extra challenge, since it entails approximating the ratio of values from large vectors, which is known to require a large amount of space to do accurately [9]. Instead, we use a slightly weaker notion of approximate deltoids to make our guarantees, and in our experimental work we will show that this approach is still highly accurate. In order to find items with large relative difference, we need to transform one of the streams. Thus this method does not work in the general cash register model, but requires that one of the streams be aggregated. This still permits the tests for one of the streams to be computed on live data as it arrives, and deltoids found between this stream and any one for which the tests have been pre-computed. Let $S_{1/k}$ be the stream whose ith entry is $S_{1/k}[i] = \frac{1}{S_k[i]}$. Then, finding items with large relative difference means finding an item i so that $D[i] = S_j[i] * S_{1/k}[i]$ is large, relative to $\sum_i D[i]$. We shall refer to this "inverted" stream by $S_{1/k}$ or just $1/k$. We choose $g = \frac{2}{\epsilon}$ and $v = \frac{4}{\epsilon}$ as for absolute differences and set

$$T_{a,b,c}(j) = \sum_{i \in G_{a,b} \cap B_c} S_j[i] \;;\; T_{a,b,c}(1/k) = \sum_{i \in G_{a,b} \cap B_c} S_{1/k}[i].$$

Tests are combined by $T_{a,b,c}(j, 1/k) = T_{a,b,c}(j) * T_{a,b,c}(1/k)$

$$|T_{a,b,c}(j,1/k)| = 1 \iff T_{a,b,c}(j,1/k) > \phi \sum_i \frac{S_j[i]}{S_k[i]}.$$

$T_{a,b,c}(j)$ is the same as in the absolute deltoid case, and so is easy to compute and maintain as new values are seen in the stream. Here, our notion of deltoids is slightly weaker: we set i to be an ϵ-approximate ϕ-deltoid by the rules:

$$D[i] \geq \phi(\textstyle\sum_i D[i]) + \epsilon||S_j||_1||S_{1/k}||_1 \Rightarrow i \in Deltoids$$
$$D[i] \leq \phi(\textstyle\sum_i D[i]) - \epsilon||S_j||_1||S_{1/k}||_1 \Rightarrow i \notin Deltoids$$

Lemma 5: $i \in Deltoids \Rightarrow$

$\forall a, c : \mathsf{Pr}[\quad (|T_{a,h_a(i),c}(j,1/k) = 1 \iff i \in B_c)]$
$\wedge \quad (|T'_{a,h_a(i),c}(j,1/k) = 1 \iff i \notin B_c)] \geq \frac{1}{2}$

Proof: For any a, c and $i \in Deltoids$, let $b = h_a(i)$. Again, assume $i \in B_c$, since the other case is symmetric. With absolute certainty,

$$\begin{aligned} i \in B_c \Rightarrow \quad & T_{a,b,c}(j) * T_{a,b,c}(1/k) \\ = \quad & (\textstyle\sum_{x \in G_{a,b}} S_j[x])(\sum_{x \in G_{a,b}} \frac{1}{S_k[x]}) \\ \geq \quad & \frac{S_j[i]}{S_k[i]} = D[i] \geq \phi \textstyle\sum_i D[i] \Rightarrow |T_{a,b,c}(j,1/k)| = 1 \end{aligned}$$

However, we also need to show that with constant probability $|T'_{a,b,c}(j,1/k)| = 0$, which is a little more involved.

$$i \in B_c \wedge (T_{a,b,0}(j,1/k) - S_j[i]/S_k[i]) < \epsilon||S_j||_1||S_{1/k}||_1 \quad (1)$$

$$\begin{aligned} \Rightarrow \quad & T'_{a,b,c}(j,1/k) = (\textstyle\sum_{x \in G_{a,b}/B_c} S_j[x])(\sum_{x \in G_{a,b}/B_c} S_{1/k}[x]) \\ \leq \quad & (\textstyle\sum_{x \in G_{a,b}} S_j[x])(\sum_{x \in G_{a,b}} S_{1/k}[x]) - S_j[i]/S_k[i] \\ < \quad & \epsilon||S_j||_1||S_{1/k}||_1 \Rightarrow |T'_{a,b,c}| = 0 \end{aligned}$$

So, we show that the probability of (1) is bounded by a constant. As before, we define some variables:

$$I_{i,x} = 1 \iff h_a(i) = h_a(x); I_{i,x} = 0 \iff h_a(i) \neq h_a(x)$$
$$Y_i = T_{a,b,0}(j,1/k) - S_j[i]/S_k[i]$$

$$\begin{aligned}\mathsf{E}(Y_i) = &\ \mathsf{E}((S_j[i] + \textstyle\sum_x I_{i,x} S_j[x]) * \\ &\ (1/S_k[i] + \textstyle\sum_y I_{i,y} 1/S_k[y]) - S_j[i]/S_k[i]) \\ \leq &\ \tfrac{1}{g} ||S_j||_1 ||S_{1/k}||_1.\end{aligned}$$

Then the probability of (1) is given by

$$\mathsf{Pr}[Y_i < \epsilon ||S_j||_1 ||S_{1/k}||_1)] > \frac{\mathsf{E}(Y_i)}{\epsilon} ||S_j||_1 ||S_{1/k}||_1 = \frac{1}{\epsilon g}.$$

Since $\epsilon g = 2$, the result follows. ■

As in previous cases, the fact that each deltoid is in $\log \frac{1}{\delta}$ groups, the overall probability of finding it is $1 - \delta$.

Lemma 6: $(i \in Deltoids \Rightarrow |V_{a,b}| = 1) \wedge (i \notin Deltoids \Rightarrow \mathsf{Pr}[|V_{a,b}| = 1] \leq \frac{1}{8})$.

Proof: The first part follows immediately from the proof of the previous lemma. For the second part, we use the same variable Y_i from the above proof. Then

$$\begin{aligned} i \notin Deltoids \Rightarrow &\ Pr[|V_{a,b}| = 1] \\ = &\ \mathsf{Pr}[Y_i + S_j[i]/S_k[i] > \phi \textstyle\sum_i D[i]] \\ \leq &\ \mathsf{Pr}[Y_i > \epsilon ||S_j||_1 ||S_{1/k}||_1] \leq \tfrac{1}{\epsilon v} = \tfrac{1}{8}\end{aligned}$$

using the Markov argument above. ■

Computing a Threshold. The threshold is $\phi \sum_i D[i]$, which can also be approximated using sketch computations [3], [11].

Theorem 4: Each ϵ approximate relative deltoid is found by the above algorithm with probability at least $1 - \delta$. The space required for finding absolute deltoids is $O(\frac{1}{\epsilon} \log(n) \log \frac{1}{\delta})$. The time to update the tests is $O(\log(n) \log \frac{1}{\delta})$ per item in the stream, and the expected time to find deltoids is $O(\frac{1}{\epsilon} \log(n) \log \frac{1}{\delta})$.

V. Practical Implementation and Study

We implemented our methods in C and conducted a set of experiments on a number of data sets, varying the parameters ϵ and δ. We also implemented two of the alternate "naive" solutions described in Section II-A, the sampling solution, and the heuristic of storing the heavy hitters (most frequent items) from each stream, and computing deltoids based on the difference between the heavy hitters for each stream. For our experiments we also exhaustively computed the "exact" deltoids for each data set, so that the output of our approximate methods could be compared to this and evaluated. To make an evaluation of the results, we computed the standard measures of *precision* and *recall* of exact ϕ-deltoids: precision is the fraction of the items returned that were correct, giving a measure of the number of false positives, and recall is the fraction of the exact deltoids that were found, giving a measure of the number of false negatives. These measures range from 0 to 1, and ideally they should both be 1. The experiments were conducted on a lightly loaded Wintel 2.4GHz machine with 512Mb of RAM.

The data sets we analyzed were drawn from a variety of network scenarios:

- **lbl-conn7** is a data set obtained from the Internet Traffic Archive [1], [30]. It consists of a record of wide-area connections taken over a thirty day period, totaling three-quarters of a million records. To study absolute and relative differences, we split the trace into two pieces covering the first and second halves of the time period. For variations, we split it into seven equal sized periods. We looked at differences in the number of bytes going to destination IP addresses.
- **phone** is a stream of 2.2 million phone calls which were captured during a single afternoon. We also split this stream in the middle to examine the difference in calling patterns in the first half compared to the second. In order to partially anonymize the data, only the area code and local exchange of the caller and destination (eg 212-555) were retained. This has the effect of aggregating over local areas. We looked at differences in the number of calls between pairs of exchanges.
- **snmp** consists of two streams of SNMP data recording all traffic over two related links in an eight day period. We compared the absolute and relative differences between the traffic sent on the links, and the variational differences within each link over the same time periods on each of the eight days. Even though the rate of generation of SNMP data is much smaller than packet or flow records, nevertheless, it is useful to see the deltoids in this data source over time.

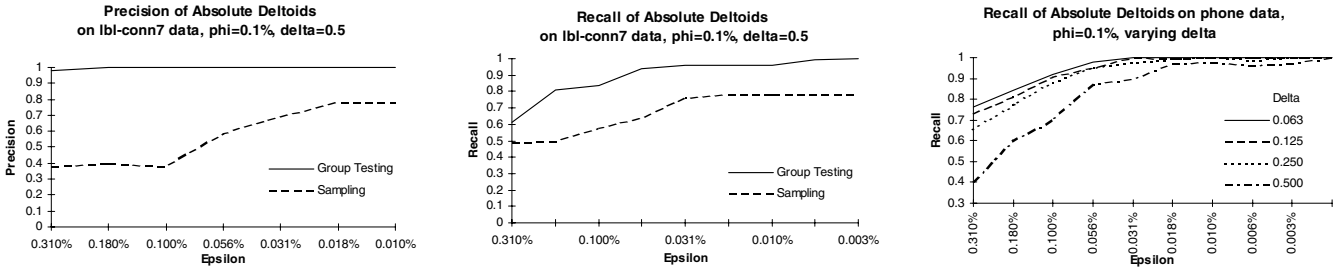
A. Experiments with Standard Approaches

When we tested the quality of sampling and then computing on the sampled data, we found that if the sampling rate was large, say sampling and storing each update with probability $\frac{1}{5}$ to $\frac{1}{10}$, then sampling gave a good approximation of the correct answer. However, the group testing method stores an amount of space that is essentially constant—it depends only on the parameters ϵ and δ, and not on the size of the stream. In order to make a fair comparison between our method and sampling, we computed the space used by our method, and set the sampling frequency to be such that the space used by the sample was the same as our method. So we have plotted the precision and recall of our method for varying ϵ, and plotted the results for sampling with the same space at corresponding x values. In each step, we multiply ϵ by a constant factor. By doing this, we see that sampling is generally inferior to group testing given equivalent space.

We also experimented with heuristics based on keeping only the Heavy Hitters in each stream and basing the deltoids on these. We omit full details of our experiments with this heuristic for lack of space: in summary, we found the heuristic to be highly unreliable, with overall poor precision and recall.

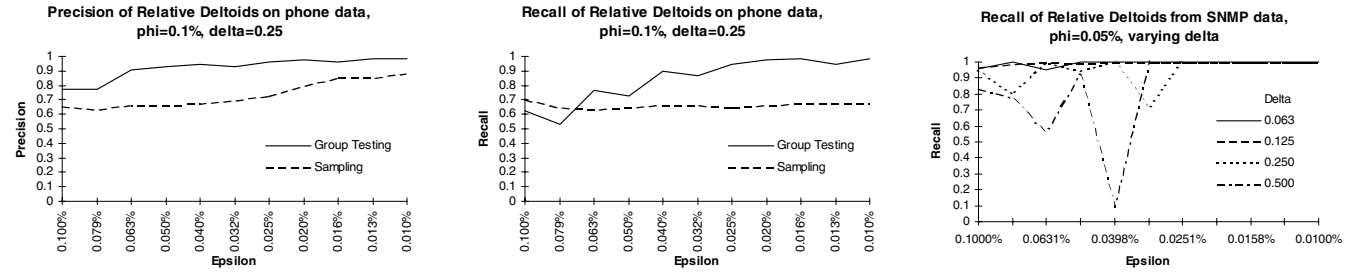
B. Group Testing for Absolute Deltoids

We conducted several experiments to determine the right settings of parameters ϵ and δ to balance accuracy with time and space consumption. We discovered that our group testing method significantly outperformed the *a priori* worst case guarantees given in Section IV. In particular, we found that the system output very few false positives even with the Verification stage bypassed. We also found that we could set ϵ and δ to quite high values and still achieve near perfect precision and recall.



(a) (b) (c)

Fig. 2. Experiments on finding Absolute Deltoids



(a) (b) (c)

Fig. 3. Experiments on finding Relative Deltoids

Figures 2 (a) and (b) show the precision and recall on lbl-conn7. In our experiments on all data sources we found that there were almost no items whose difference consumed a very large fraction of the total difference (say, 10%). The largest few deltoids have difference around 5%, and there are typically around twenty in the range 1% to 0.5%. For our experiments, we set the threshold ϕ to be 0.1%, meaning there were between 100 and 200 deltoids. Interestingly, many of the absolute difference deltoids, the largest few included, were items whose difference was between a moderately large item in the first stream and a larger value in the second, meaning that they were distinct from the relative difference deltoids. As in most of our experiments, group testing achieved near perfect precision throughout, with little variation. Recall improves as ϵ is shrunk, and reaches the optimal value around $\epsilon = \frac{\phi}{10}$. Figure 2 (c) shows the effect of varying δ on recall for phone data (precision was 1 throughout). We see that although decreasing δ always improves recall, beyond $\delta = 0.25$ the effect is very small, meaning that it suffices to set $\delta = 0.25$, corresponding to two copies of the identification test.

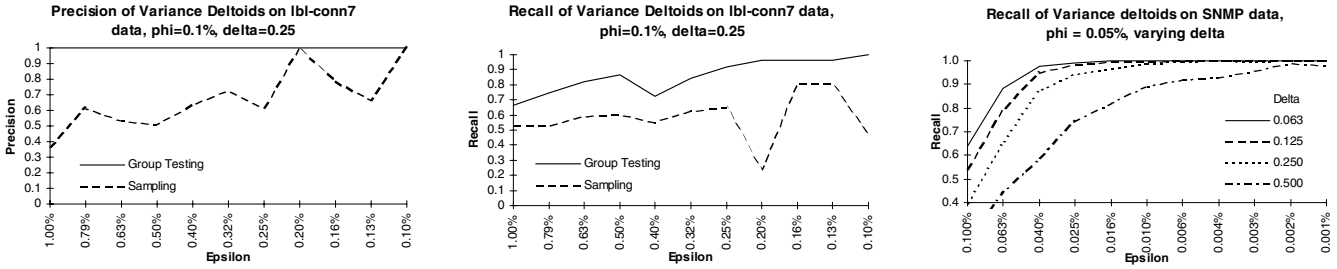
C. Group Testing for Relative Deltoids

Finding relative difference deltoids turned out to be the most challenging problem. Setting the right value of ϕ is important here: set ϕ too low and everything is a deltoid, set it too high and there are no deltoids. It is therefore an important feature of our method that ϕ can be specified at query time: only ϵ needs to be chosen in advance. The relative difference deltoids were items which were moderately large in the first stream, but whose count had dropped to zero or single digit figures in the second stream. This makes them challenging to find. In Figure 3 (a) and (b) we see that Group Testing outperforms sampling over most settings of ϵ. Acceptable results are obtained when $\epsilon = \frac{\phi}{3}$, and perfect results by the time $\epsilon = \frac{\phi}{10}$.

Figure 2 (c) shows the difficulties with certain data sets: on SNMP data, if ϵ is not set low enough, then the recall is highly variable, meaning that many deltoids are missed. A lower δ helps somewhat, and the phenomenon disappears when $\epsilon < \frac{\phi}{2}$, meaning that it is vital to know approximate upper bounds on ϕ for the traffic source of interest. In all our experiments, we found that $\phi = 0.1\%$ or 0.05% covered the top two hundred deltoids; more than this is unlikely to be informative, and already this is stretching the amount of information a network manager will want to see.

D. Group Testing for Variational Deltoids

The results for variational deltoids are shown in Figure 4. Here, group testing performed very well: good results were achieved by setting $\epsilon > \phi$. We conjecture that this is partly due to the way in which variational deltoids are defined: because they are based on the square of the deviation from the mean, this means that deltoids have a significantly larger difference than non-deltoids (as contrasted with the relational case, where we found that the difference of deltoids was not much different to the difference of non-deltoids, contributing to the difficulty of getting perfect precision). Optimal results were achieved on the lbl-conn7 data set by setting $\epsilon = \phi$. Figure 4 (c) shows



(a) (b) (c)

Fig. 4. Experiments on finding Variational Deltoids

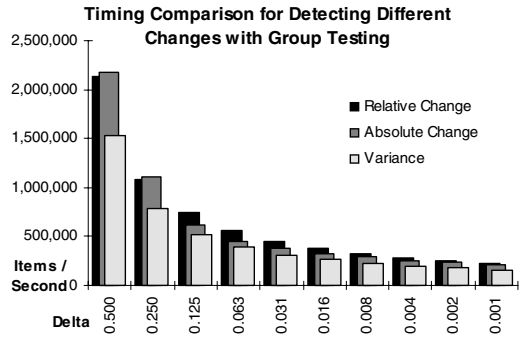


Fig. 5. Processing Rate of different methods as δ varies

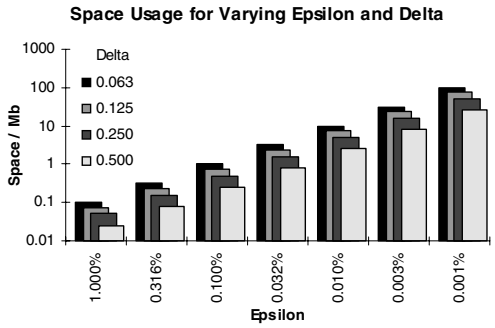


Fig. 6. Space Usage as ϵ Varies for Different Values of δ

that for variational too, recall is improved by decreasing δ, but that even for $\delta = 0.25$ then optimal recall is achieved for a modest value of ϵ relative to ϕ.

E. Space and Time Costs

We ran speed trials to determine whether our methods were capable of operating at network line speeds. The results were very encouraging. Our code was not optimized, and included several routines for checking and supporting output for the experiments, so we believe that an optimized implementation running on dedicated hardware could improve the throughput a lot. For each method, we computed how many items per second the method could process (here, the items were taken to be 32 bit IP addresses and packet sizes of traffic from each address). The results are shown in Figure 5. We study the effect of varying δ on the item processing cost: note that ϵ does not factor in the update time, only in the space and query costs.

As expected, absolute and relative differences take about the same time, since the update algorithms are almost identical. For variational deltoids, we need to compute an additional 4-wise independent hash function; however, this additional computation does not seem to have a disastrous impact in the processing speed, and reduces the packet processing right by an average of about a third. Since in our earlier experiments, we saw that setting $\delta = 0.5$ gives high output quality, then we benchmark our system as capable of processing around 1 million items per seconds. This means that it is easily capable of processing traffic rates on 100Mbs links, and with some work then 1Gbps and higher are within reach.

The space usage was also reasonable. Figure 6 shows how the space needed varies as a function of ϵ and δ. In our previous experiments, we determined that the very highest difference deltoids occur around $\phi = 5\%$, and so can be found with very small space—say, around 10k. For the top ten or twenty deltoids, then setting $\phi = 0.5\%$ sufficed, meaning we need around 100k to find them. To find the top one hundred to two hundred deltoids, this gives a space requirement of between 500k and at most 2 or 3Mb per stream. In practice, a network manager will only want to see the very highest deltoids, or those which consume more than a small fraction of the total bandwidth.

VI. Extensions

We consider a number of ways in which our work can be extended. Our ongoing work is to experimentally study our algorithmic these extensions.

- **Comparing Different Time Windows, Speeds, Granularity and Predictions.** Throughout this work, we have assumed that pairs of streams represent the same traffic volume, so that values for each item are comparable. But we would also like to be able to compare, say, the traffic in the last hour to the traffic in the last week, or the traffic on a fast link to the traffic on a much slower link. The solution is to *scale* all traffic linearly

so that the two streams have the same scaled traffic. An important consequence of the linearity of the tests in our algorithms is that such scaling by α can be done by scaling all values stored in the tests by α. Similarly, one can take our data structure for the interfaces and add them to consider the total traffic per router or take that for each hour and add them up to consider total traffic per day, etc. because of this linearity; hence, our methods work for different granularities. This also allows a wide variety of predictive models to be tested. Comparing the last hour to the current hour can be thought of a prediction that subsequent hours should look similar. The deltoids are the items which are behaving differently to how they are predicted. Other prediction models—say, an average of the last 24 hours, or an exponentially weighted average—can be made by making the appropriate linear combination of tests for the past data.

- **Faster Implementation.** Our current implementation is fairly fast, but there are some improvements that may speed up the stream processing. First, we observed in the course of our experiments that sampling at a sufficiently high rate (say, 10 - 20%) preserves most of the deltoids. (The same is not true when we sampled to 1% or lower.) This suggests that if we first sample the stream as it arrives, and pass only the sampled items to the group testing, then this should still find most deltoids, while increasing the capacity of the system by a factor of 5–10. Another direction is to try to speed up the update procedure itself. One reason that it is slow is that it considers a bit at a time of the index of the item. At the cost of increasing the space used, we could consider larger divisions: say, consider the index a byte at a time, and keep 256 tests—one for each byte value. This increases the space needed by a large factor—we need 256 tests per byte of the index, instead of 8—but results in a theoretical speed up of a factor of 8.

There are a number of possible extensions that are not fully answered; for example, we have focused on solving the fundamental question of how to find and detect single items which exhibit difference. Our methods extend to when the items are *multidimensional* (so consider source and destination address, instead of source or destination), or when they are arranged into a hierarchy (such as network/subnet/address). But *hierarchical* data such as the IP universe presents a new challenge: here, the aim is to find *prefix* deltoids which consume ϕ of the total difference after the contribution of any deltoids that share this prefix have been discounted. A version of group testing in order to find hierarchical heavy hitters is described in [8], the deltoid problem on hierarchies is yet to be solved.

VII. Related Work

There has been some recent work on finding various deltoids. A significant contribution [25] propose a set of methods to find changes in network data, which fall into the category we term "Sketch-based methods" in Section II-B. Much of their work is complementary to ours, since they propose a variety of "prediction methods" to compare the observed counts against predictions based on past observations. When there is a significant (absolute) difference between the count and the predicted count, the item is output. Throughout, we have assumed the simplest prediction model: comparing the value of each item to its value in another time period or in another location. But the prediction methods proposed in [25] are based on linear transformations of previous readings (average, weighted average, etc.) and so can be applied here by performing the same transformation on the values stored in the test data structures. However, a different method is proposed for finding the items that are changed, based on building sketches of the data as it is observed. As we pointed out in Section II-B, this limits the applicability of this approach. While deltoids can be found online by querying the sketch every time an item arrives, it is not possible to use this approach offline: after a stream has been seen, to compare it to the summaries of several other streams, and find deltoids between each (or linear combinations of each). This shows the strength of the group testing approach: deltoids can be found in both the online scenario and the offline one. The methods we have described here not only solve the absolute difference problem considered in [25], but also apply to the relative and variational cases. Similarly, in [6], the authors considered the problem of finding absolute deltoids, but their method took two passes over the data. In contrast, our result here finds absolute deltoids in one pass. The authors in [6] explicitly left finding relative deltoids on data streams as an open question; this problem is also explicitly stated in the context of web search traffic as an open problem in [22] where it is called the problem of top gainers and losers. In this paper, we give the first solution to the problem of finding relative deltoids.

The problem of finding absolute deltoids was also studied in a recent paper [14] where the authors consider reporting "compressed deltas" which may be thought of as "hierarchical absolute deltoids" from Section VI. The authors propose algorithms based on finding heavy hitters in each stream and using that to prune the search space for finding absolute deltoids. The pruning is done either in multiple passes, or by using the candidates from one stream to search the other. These approaches do not give a provable guarantee on the quality of absolute deltoids that are reported, as we are able to do. However, [14] highlights the challenges of network traffic data analyses we address, and a good discussion of the issues and difficulties.[4]

A number of results are known which are somewhat related to ours. For example, various norm aggregates of differences have been studied in the data stream context including L_1 norm [18], Hamming norm [7], etc. These methods provide estimates of the norm, say sum total of the differences, but do not explicitly determine the items that have large differences.

Combinatorial group testing (CGT) has a long history [13]

[4]This paper discusses "relative" deltoids, but relative to them means scaling each data stream by a scalar, so that notion is different from our definition of relative deltoids.

because it has myriad applications. CGT is inherent in small space algorithms from learning theory [26] as well as data stream algorithms for finding histograms and wavelets [20]. The problem of finding heavy hitters was addressed in [15] where an item was a heavy hitter if it exceeds a fixed threshold. More recently, we used CGT for finding heavy hitters in data streams [10] for database scenarios where items are inserted and deleted. Our work here extends this approach substantially by introducing different group tests to find different deltoids, thereby deriving powerful new results as well as making it a general framework with many applications.

The area of data streams—designing algorithms that use small space, handle updates rapidly, and estimating different quantities of interest—has become popular in database research, networking and algorithms. There have been tutorials [19], [31], workshops [2] and surveys [5], [29]. Our results add to the growing set of techniques in this area.

VIII. Concluding Remarks

We initiated the study of finding significant differences in network data streams, so that network managers can be kept up to date with "what's new". Our methods require small amounts of memory and operate very quickly, able to process millions of records per second on a standard desktop computer. Our solutions are all based on a structure of Combinatorial Group Testing, which gives a flexible framework for detecting any kind of difference, given a suitable test definition. The structure can be used to find absolute, relative and variational differences, between traffic in different time periods, interfaces or routers. Different link speeds can be compensated for, different prediction models used, and there are prospects for pushing the data rate to hundreds of millions of packets per second. The result is a scalable, effective method for monitoring and analyzing traffic usage patterns as part of an ongoing network management task.

References

[1] Internet traffic archive. `http://ita.ee.lbl.gov/`.

[2] Workshop on management and processing of data streams (MPDS), 2003.

[3] N. Alon, P. Gibbons, Y. Matias, and M. Szegedy. Tracking join and self-join sizes in limited storage. In *Proceedings of the Eighteenth ACM Symposium on Principles of Database Systems (PODS '99)*, pages 10–20, 1999.

[4] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 20–29, 1996. Journal version in *Journal of Computer and System Sciences*, 58:137–147, 1999.

[5] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of Symposium on Principles of Database Systems (PODS)*, pages 1–16, 2002.

[6] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Procedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, pages 693–703, 2002.

[7] G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan. Comparing data streams using Hamming norms. In *Proceedings of 28th International Conference on Very Large Data Bases*, pages 335–345, 2002. Journal version in *IEEE Transactions on Knowledge and Data Engineering* 15(3):529–541, 2003.

[8] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Finding hierarchical heavy hitters in data streams. In *International Conference on Very Large Databases*, pages 464–475, 2003.

[9] G. Cormode and S. Muthukrishnan. Estimating dominance norms of multiple data streams. In *Proceedings of the 11th European Symposium on Algorithms (ESA)*, volume 2838 of *LNCS*, 2003.

[10] G. Cormode and S. Muthukrishnan. What's hot and what's not: Tracking most frequent items dynamically. In *Proceedings of ACM Principles of Database Systems*, pages 296–306, 2003.

[11] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 2004. in press.

[12] M. Datar and S. Muthukrishnan. Estimating rarity and similarity over data stream windows. In *Proceedings of 10th Annual European Symposium on Algorithms*, volume 2461 of *Lecture Notes in Computer Science*, pages 323–334, 2002.

[13] D-Z Du and F.K. Hwang. *Combinatorial Group Testing and Its Applications*, volume 3 of *Series on Applied Mathematics*. World Scientific, 1993.

[14] C. Estan, S. Savage, and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. In *Proceedings of ACM SIGCOMM*, 2003.

[15] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proceedings of ACM SIGCOMM*, volume 32, 4 of *Computer Communication Review*, pages 323–338, 2002.

[16] C. Estan and G. Varghese. Data streaming in computer networks. In *Proceedings of Workshop on Management and Processing of Data Streams*, `http://www.research.att.com/conf/mpds2003/schedule/estanV.ps`, 2003.

[17] C. Estan, G. Varghese, and M. Fisk. Bitmap algorithms for counting active flows on high speed links. Technical Report CS2003-0738, UCSD, 2003.

[18] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate L_1-difference algorithm for massive data streams. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, pages 501–511, 1999.

[19] M. Garofalakis, J. Gehrke, and R. Rastogi. Querying and mining data streams: You only get one look. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2002.

[20] A. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, pages 389–398, 2002.

[21] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. QuickSAND: Quick summary and analysis of network data. Technical Report 2001-43, DIMACS, 2001.

[22] M. Henzinger. Algorithmic challenges in search engines. *Internet Mathematics*, 1(1):115–126, 2003.

[23] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proceedings of the 40th Symposium on Foundations of Computer Science*, pages 189–197, 2000.

[24] R. Karp, C. Papadimitriou, and S. Shenker. A simple algorithm for finding frequent elements in sets and bags. *ACM Transactions on Database Systems*, 2003.

[25] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based change detection: Methods, evaluation and applications, 2003. manuscript.

[26] E. Kushilevitz and Y. Mansour. Learning decision trees using the fourier spectrum. *SIAM Journal on Computing*, 22(6):1331–1348, 1993.

[27] G.S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of 28th International Conference on Very Large Data Bases*, pages 346–357, 2002.

[28] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[29] S. Muthukrishnan. Data streams: Algorithms and applications. In *ACM-SIAM Symposium on Discrete Algorithms*, `http://athos.rutgers.edu/~muthu/stream-1-1.ps`, 2003.

[30] V. Paxson. Empirically derived analytic models of wide-area tcp connections. *IEEE ACM Transactions on Networking*, 2(4):316–336, 1994.

[31] G. Varghese. Detecting packet patterns at high speeds. Tutorial at SIGCOMM 2002.

[32] V. Yegneswaran, P. Barford, and J. Ullrich. Internet intrusions: global characteristics and prevalence. In *Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS-03)*, volume 31, 1 of *Performance Evaluation Review*, pages 138–147, 2003.